Evolved and Timed Ants:

Optimizing the Parameters of a Time-Based Ant System Approach to the Traveling Salesman Problem Using a Genetic Algorithm.

Damon Cook

New Mexico State University

Computer Science Department

August 4, 200O

Sandia National Laboratories

Department 6238 : Software Engineering

Manager : Michael Tebo

Technical Advisor : David Harris

Abstract – This project uses a Genetic Algorithm to optimize the Ant System approach to the Traveling Salesman Problem. The Ant System approach, practitioned by E. Bonabeau, M. Dorigo and G. Theraulaz is modeled after the behavior of an ant colony. The Genetic Algorithm that I am using is similar to those discussed by D. Goldberg. In addition to the elements discussed by Bonabeau, et al., this program uses a time-based system to more closely resemble a biological ant colony. This time-based system more closely mimics its biological counterparts by basing the movements of the ants on the relative time taken between cities. The results indicate that this system is accurate for small problems and also reasonably accurate for medium problems.

1. Introduction

When solving search problems with computers, a common approach is to calculate every possible solution and then choose the best of those as the answer. Unfortunately, some problems have such large solution spaces that this is impossible to do. These are problems where the solution set grows exponentially with the amount of inputs. These problems are referred to as n-p hard or n-p complete problems. One such problem is The Traveling Salesman Problem (TSP). This paper discusses a method for solving the TSP that is based on the natural foraging system of an ant colony. This Ant Colony (AC) approach is a kind of educated guess system. Random solutions are tried and then better solutions are found based on the results of the earlier tries. This is a way of "optimizing" a solution, as opposed to "solving for" the solution. Of course, the AC also has several inputs. The question of which are the best inputs is another problem that can be optimized. In this case, we used a Genetic Algorithm (GA), another optimizing approach, to optimize these parameters.

1.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a problem taken from a real life analogy. Consider a salesman who needs to visit many cities for his job. Naturally, he would want to take the shortest route through all the cities. The problem is to find this shortest route without taking years of computation time. Originally the TSP was presented as a contest with a path through only a few cities. The first thought that would come to any programmer's mind when presented with the TSP would be to check every possible route and take the shortest one as the answer. Unfortunately, the number of routes grows exceptionally fast. If this problem only needed to find the route with 10 cities, it would be manageable. If there were 100 cities, the number of possible routes would have over 150 digits. Obviously, this method quickly becomes unfeasible. Today, solutions have been proposed that can solve for as many as 13,509 cities (this was done with a custom algorithm and may have taken as long as four years; see [14] for more information).

The applications of the TSP are fairly numerous. It can be applied to anything from drilling holes in printed circuit boards to designing fiber-optic communications networks to coordinating military maneuvers to routing helicopters around oil rigs.

There have been several types of approaches taken to solving the TSP. They include:

- Heuristic approaches [1, 2]
- Memetic algorithms [6, 12]
- Ant colony optimizations [7, 11]
- Simulated annealing [3, 9]
- Genetic algorithms [5, 10]
- Neural networks [4]
- And various other methods for more specific variations of the TSP.

These approaches do not always find the true optimal solution. Instead, they will often consistently find good solutions to the problem. These good solutions are typically considered to be optimal simply because they are the best that can be found. This is what is often actually meant by the term optimal.

1.2 Ant Colony Optimization

Ant Colony or Ant System optimizations are based on the natural foraging system found in an ant colony. For example, we've all seen how ants will make a long line from their anthill to a food source. Biologists have found that this is done using a pheromone (a chemical that other ants can detect easily) trail left by each ant. When an ant finds a food source, its pheromone trail will lead other ants to that same food source. In just a short time, there will be a large line of ants moving back and forth between the food and the anthill. If the food source is removed, the ants will continue to follow the trail for a time but will eventually move off in a more random pattern until they find another food source.

Similarly to ants forming a line to a food source, programmed "ants" can be used to form a line through the shortest path of the TSP. The "ants" leave behind some amount of "pheromone" wherever it goes. When another "ant" needs to decide which path to take, the "pheromone" on the trail will influence it to take the same path as the previous ant.

Simply leaving pheromone on a path is not enough to get the optimal solution. Obviously, the "ants" cannot simply follow the same paths every time. Other factors need to be worked in. The "ant" cannot simply take the path with the most pheromone; it must have a chance to randomly choose a new path to find a better solution to the problem. This is done by applying simple probability mathematics to the decision that the "ant" will make. The more "pheromone", the more likely the ant is to choose the path, but there is still a chance that it will take a different path. Additionally, the "pheromone" can have an "evaporation rate" which would also add to the probability that an "ant" would find a newer, better path.

The main goal is to modify the behavior of individual ants to produce a desired response in the colony behavior. This is done with the use of several parameters. These parameters were optimized using a Genetic Algorithm (GA).

1.3 Genetic Algorithms

A genetic algorithm is based on the same idea as the theory of evolution. Basically, several random sets of parameters are applied to an algorithm and a fitness value is returned for each. Based on these fitness values, the best sets are mixed together and new sets are again applied to the algorithm until an optimal set of parameters is obtained. This effect is usually obtained by breaking the genetic algorithm into a few small parts. The main parts are the fitness function and the evolution function.

The evolution function produces a string of inputs (often a string of bits that are encodings of the input parameters) then asks the fitness function for a fitness value for that string. When several strings have been assigned a fitness value, the evolution function takes the best strings, mixes them together, sometimes throws in a "mutation" to the strings and then sends the results back as new input strings.

The fitness function of a genetic algorithm takes in a string of inputs and runs them through the process that is being evaluated. Based on the performance of the inputs, the function returns a fitness value. In the case of the TSP, the fitness function returned the total length or weight of the path found.

2. Some Earlier Approaches to Solving the Traveling Salesman Problem

There have been many approaches to solving the Traveling Salesman Problem (TSP). These approaches range from a simple heuristic algorithm to algorithms based on the physical workings

of the human mind to those based on ant colonies. These algorithms all have the same ultimate goal: in a graph with weighted edges, find the shortest Hamiltonian path (the path with the smallest sum of edge weights). Unfortunately, this goal is very hard to achieve. The algorithms therefore settle for trying to accomplish two smaller goals: (1) To more quickly find a good solution and (2) To find a better good solution. A good solution is one that is close to being optimal and the best of these good solutions is, of course, the optimal solution itself.

2.1 Heuristic Algorithms and the TSP

The word 'heuristic' means "A rule of thumb, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike algorithms, heuristics do not guarantee optimal, or even feasible, solutions and are often used with no theoretical guarantee." In contrast, an algorithm is defined as "a precise rule (or set of rules) specifying how to solve some problem." To combine these together into a heuristic algorithm, we would have something like "a set of rules specifying how to solve some problem by applying a simplification that reduces the amount of solutions checked". In other words, the algorithm is the instructions for choosing the correct solution to the problem while the heuristic is the idea of how to shrink the list of possible solutions down to a reasonable size.

An example of a heuristic approach to the TSP might be to remove the most weighted edge from each node to reduce the size of the problem. The programmer in this situation may assume that the best solution would not have the most weighted edge. Upon close inspection, this heuristic may not actually give the best solution, maybe not even a feasible solution (if all of the most weighted edges from each node are connected with the same node) but it may be a calculated risk that the programmer takes.

The main idea of a heuristic approach to a problem is that, although there is exponential growth in the number of possible solutions to the problem, evaluating how good a solution is can be done in polynomial time.

In dealing with the TSP, the most common uses of heuristic ideas work with a local search. Similarly to the above example, the heuristic does not try to encompass every possibility of the problem at hand; instead it attempts to apply common sense to shrink the problem to a manageable size.

Perhaps the most-used local search heuristic that is applied to the TSP is the n-opt method developed by Lin and Kernighan [2]. It simply takes a random path and replaces n edges in it until it finds the best of those paths. This is typically done where n is set to 2 or 3 [1]. These methods were applied to several different problems. Notably, they were able to find the optimal solutions for a 42-city problem 4 out of 10 times and the optimal solution to a 48-city problem 2 out of 10 times [1] (the 10 times in these were running concurrently so the optimum solution was found in each run of the program).

2.2 Simulated Annealing and the TSP

Simulated Annealing is a method that is based on the cooling of a physical system. The general idea is that there is a temperature (T) and a cost function (H). In our case, the cost function is the sum of the weights of the edges in our circuit. In the beginning, there is a random solution to the problem. At each iteration, a change is proposed to this solution and that change is evaluated based on the cost function and the temperature. If the cost function decreases then the change is accepted. If the cost function does not decrease then the change is accepted or rejected based on the temperature. The higher the temperature, the better the chance that the change will be

accepted. As time progresses, the temperature decreases and eventually there is no possibility for a change to occur without the cost function decreasing. This was first described by Metropolis, et al. [3]. Using this method, researchers such as Lin, et al. were able to get to within two units of the optimal cost for problems up to a size of 100 [9].

2.3 Neural Networks and the TSP

According to Haykin, S. (1994), Neural Networks: A Comprehensive Foundation, NY: Macmillan, p. 2:

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge."

Basically, a neural network (NN) is made up of many independent units (neurons) and connections between them. The connections are given various weights based on a "learning process". Based on the sum of the products of adjoining neurons and the weights of the connecting edges, each neuron finds a value. Additionally, if the value of one neuron changes then the values of all the adjoining neurons also change. This creates a ripple effect that can change the values of every neuron (although it could also change none of them).

An NN can be applied to a TSP with n cities. This is done by creating n^2 neurons. The output of each neuron (v_{xi}) represents whether city x is visited as the i-th city in the sequence. It is a 1 if this is true or a 0 if it is not. Additionally, the amount d_{xy} is applied to the calculations as the distance between cities x and y. This is done using various mathematical formulas that I will not list here. Some of the various implementations were reviewed by Matsuda [4].

2.4 Genetic Algorithms and the TSP

A genetic algorithm is based on the same idea as the theory of evolution. Basically, several random sets of parameters are applied to an algorithm and a fitness value is returned for each. Based on these fitness values, the best sets are mixed together and new sets are again applied to the algorithm until an optimal set of parameters is obtained. This effect is usually obtained by breaking the genetic algorithm into a few small parts. The main parts are the fitness function and the evolution function.

The evolution function produces a chromosome (an encoding of the parameters which is often represented as a string of bits) then asks the fitness function for a fitness value for that chromosome. When several chromosomes have been assigned a fitness value, the evolution function takes the best ones, mixes them together, sometimes throws in a "mutation" to the strings and then sends the results back as new chromosomes. In the case of the TSP, the parameters might be the order of the nodes through which the path must go.

The fitness function of a genetic algorithm takes in a string of inputs and runs them through the process that is being evaluated. Based on the performance of the inputs, the function returns a fitness value. In the case of the TSP, the fitness function might return the total length of the path found. Genetic algorithms and their applications to the TSP are described by Goldberg [5]. He points out that close to optimal solutions have been found using these methods for problems of up to a size of 200 by researchers such as Bonomi and Lutton [IO].

2.5 Memetic Algorithms and the TSP

A memetic algorithm (MA) is really a combination of several different techniques. Generally, an MA can be thought of as an algorithm that combines local search heuristics with a crossover operator (the same type of mixing and matching that happens with a GA's evolution function). Despite this, the difference between an MA and a GA is very distinct. As opposed to the fitness functions of GAs, MAs use a local search heuristic to determine how the parameter definitions will be modified at each iteration. For example, an MA might use simulated annealing to find a solution with some parameters and return that value to the crossover operator just like a GA would return a value from a fitness function. For this reason there are many other terms used to refer to MAs including Hybrid Genetic Algorithms, Parallel Genetic Algorithms, and Genetic Local Search Algorithms. The research in MAs was most notably conducted by Mascato [6]. Researchers such as Mühlenbein have shown MAs to be near-optimal with sizes at least as large as a 200-city problem [12].

2.6 Ant Colony Algorithms and the TSP

Ant-based algorithms are based on studies of ant colonies in nature. The main idea in these algorithms is that the behavior of each individual ant produces an emergent behavior in the colony. When applied to the TSP, individual agents ("ants") traverse the graph of the problem, leaving a chemical (pheromone) trail behind them. At each node it comes to, an ant must decide which edge to **take** to the next node. This is done by checking each edge for pheromone concentration and applying a probability function to the decision of which edge to choose. The higher the concentration of pheromone, the more likely the ant is to choose that edge. Also, to avoid stagnation in travel, the pheromone is given an evaporation rate so that at each iteration the pheromone loses a certain percentage on each edge. This method was researched originally by Dorigo, et al. [7]. This method has been shown to do better than other algorithms on random 50-city problems as well as finding the optimum solutions for problems with up to 100 cities [11].

3. Methods Used

This project combined two computer optimization methods: a variation on the Ant Colony (AC) method and the Genetic Algorithm (GA) method. These methods were combined together to optimize a solution for the Traveling Salesman Problem (TSP).

3.1 The Time-Based Ant Colony

A typical Ant Colony (AC) program would run the "ants" through the "map" of the problem in one iteration, followed by a time of dropping pheromone on various edges and then repeat. The program used here was slightly different. In an attempt to more closely model the natural order of an ant colony, a time system was introduced. The basic idea of this system is that a time loop is run as the main loop. At each iteration, the time would increase by a certain amount. Also, at each iteration, each "ant" would move further down its path. When it reached a new node, it would choose which node to travel to next while at the same time leaving some pheromone on the path it had just taken. The other difference is that after an "ant" made a complete route (through every node) it would turn around and head back the exact same way it had gone. Some basic pseudocode might be as follows:

```
for time = 0 to max_time STEP increment-value
  for i = 1 to number-of-ants
    if ( ants[i] has time left to get to the next node )
      decrement ants[i]'s time left by increment-value
    else
      pick which node to move to next, set ants[i]'s time left,
      and update the pheromone on the edge just passed
```

The decision-making process that each "ant" went through when deciding which node to move to next was loosely based on the decision-making process of the AC system described by Bonabeau, et al. [8]. Basically, a list of possible choices was formed (nodes that the "ant" had not yet visited on its route), then the "ant" had a random chance of picking the next node based on pheromone or not. If the "ant" did not choose based on pheromone, then it picked a random node from its list. If it did base its decision on pheromone, then it evaluated the nodes in its list based on their distance from the current node as well as the amount of pheromone on the path to that node as follows:

Pick next node X to satisfy the equation:

$$\max_{X \in A} \left(d_{CX}^{E} * p_{CX}^{-F}\right)$$

where C is the current node, A is the set of all unvisited nodes, d is the distance from C to X, p is the amount of pheromone on the path between C and X, E is the distance factor (dist_fact) and F is the pheromone factor (pher_fact).

Another important part of any AC system, is the evaporation of the pheromone. The pheromone can't just be left to sit forever on a path or fewer new paths will be chosen. This means that some sort of "evaporation" of the pheromone must occur. In this time-based system, the evaporation occurs at regular intervals. At these regular intervals, the pheromone would be reduced by some percentage called an "evaporation rate".

When an "ant" had gone through its whole route, and then returned to its starting point, the time it took the ant was compared to the best recorded time. If it was a better amount of time, then the length of the path it took was compared to the best recorded path length. Finally, if the path length was shorter than the best recorded, it was locally optimized and then pheromone was dropped on this locally optimized best path. The local optimization basically went through each node in the path, switched it with every other node and if it made a better path, the switch was kept. If not, it was returned and the next switch was tried. This is one of the simplest local optimization procedures that could have been used. The pseudocode for this local optimization might be as follows:

```
for i = 2 to length of Best-Path - 1
  for j = 3 to length of Best-Path
    if(i does not equal j)
      temp = get_path_weight(Best_Path);
```

```
switch(Best_Path[i] and Best_Path[j])
if(get_path_weight(Best_Path) > temp)
  switch(Best_Path[i] and Best_Path[j])
```

In this system, there were several inputs. These inputs were optimized using a Genetic Algorithm (GA) and included:

1. max time* – the amount of "time" that the "ants" explore.
2. increment – defines how many time steps happen at each iteration of the time loop.
3. num of ants -the number of "ants" that are exploring the TSP map.
4. evaporation increment – how many iterations of the time loop are between each evaporation.
5. evaporation rate – the percentage of pheromone that "evaporates" away.
6. add pheromone1 -the amount of pheromone that is added when the "ant" moves to a new node.
7. add pheromone2 -the amount of pheromone that is added when the "ant" returns along the path it came – also amount dropped on best path.
8. dist factor – relates how important the distance of a node is to the "ant".
9. pher factor – relates how important the pheromone on a path is to the "ant".
10. rand thresh -determines how likely the "ant" is to choose based on pheromone.

*max_time was not an optimized parameter, instead it varied based on the size of the map used.

3.2 The Genetic Algorithm

The Genetic Algorithm (GA) used in this program was written by D. Harris and then modified for use with the Ant Colony described above. It used a Tournament style of elite selection and a two-point crossover method. The crossover rate used was .7 and the mutation rate was .001. It was run with a population size of fifty for 100 generations. In most cases, results were found much earlier than the 100"' generation and the results section specifies what generation the numbers were taken from. The fitness is recorded as the inverse of the best path length. Finally, each "chromosome" included 70 genes that were divided among the input parameters as follows:

1. increment – 5 genes range: [1, 32]
2. num of ants – 8 genes range: [1, 256]
3. evaporation increment – 5 genes range: [1, 32]
4. evaporation rate – 8 genes range: (0, 1]
5. add pheromone1 – 8 genes range: (0, 1]
6. add pheromone2 – 8 genes range: (0, 1]
7. dist factor – 10 genes range: (0, 10]
8. pher factor – 10 genes range: (0, 10]
9. rand thresh – 10 genes range: (0, 1]

The first three numbers were actual binary representations of the numbers while the other six were found by dividing the intervals into small pieces and taking some amount of those sections as the parameter based on the binary representation of that amount.

3.3 The Traveling Salesman Problems

The TSPs used in testing this system were both found at the TSPLIB [13]. The ones used were known as ulysses1 6 and ei15 1 (16-city and 5 l-city maps respectively). Both are given by coordinates on a two dimensional Cartesian plane and the distances were calculated using the Pythagorean theorem. The ulysses1 6 map was used with a max-time value of 50000 and the ei15 1 map was used with a max-time value of 200000.

4. Results

For the map ulysses1 6, the optimal solution was found by this algorithm. That solution is of length 73.99. That path was found early on in most runs of the program. Sometimes as early as generation 6 (see Table 2).

For the map ei15 1, the optimal solution was not found by this algorithm. The optimal solution had a best path length of 429.98. The best solution found by this algorithm was actually 432.16 (see Table 1).

Parameter	Generation 14 Path Length: 432.16	Generation 28 Path Length: 438.79	Generation 35 Path Length: 435.04
increment	7	3	3
num_of_ants	172	175	175
evaporation_increment	9	9	3
evaporation_rate	0.533	0.533	0.784
add_pheromone1	0.961	0.961	0.333
add_pheromone2	0.392	0.271	0.267
dist_factor	8.856	9.756	4.594
pher_factor	5.611	0.039	0.039
rand_thresh	0.898	0.965	0.965

Table I: Parameters found during several generations of a typical run on map eil51

Parameter	Generation 6 Path Length: 73.99	Generation 20 Path Length: 73.99	Generation 21 Path Length: 73.99
increment	1	2	2
num_of_ants	255	239	239
evaporation_increment	18	16	16
evaporation_rate	0.906	0.612	0.612
add_pheromone1	0.227	0.878	0.878
add_pheromone2	0.933	0.337	0.314
dist_factor	8.436	9.032	6.530
pher_factor	8.270	2.981	3.011
rand_thresh	0.671	0.714	0.557

*Table 2: Parameters found during several generations of a typical run on map **ulysses16***

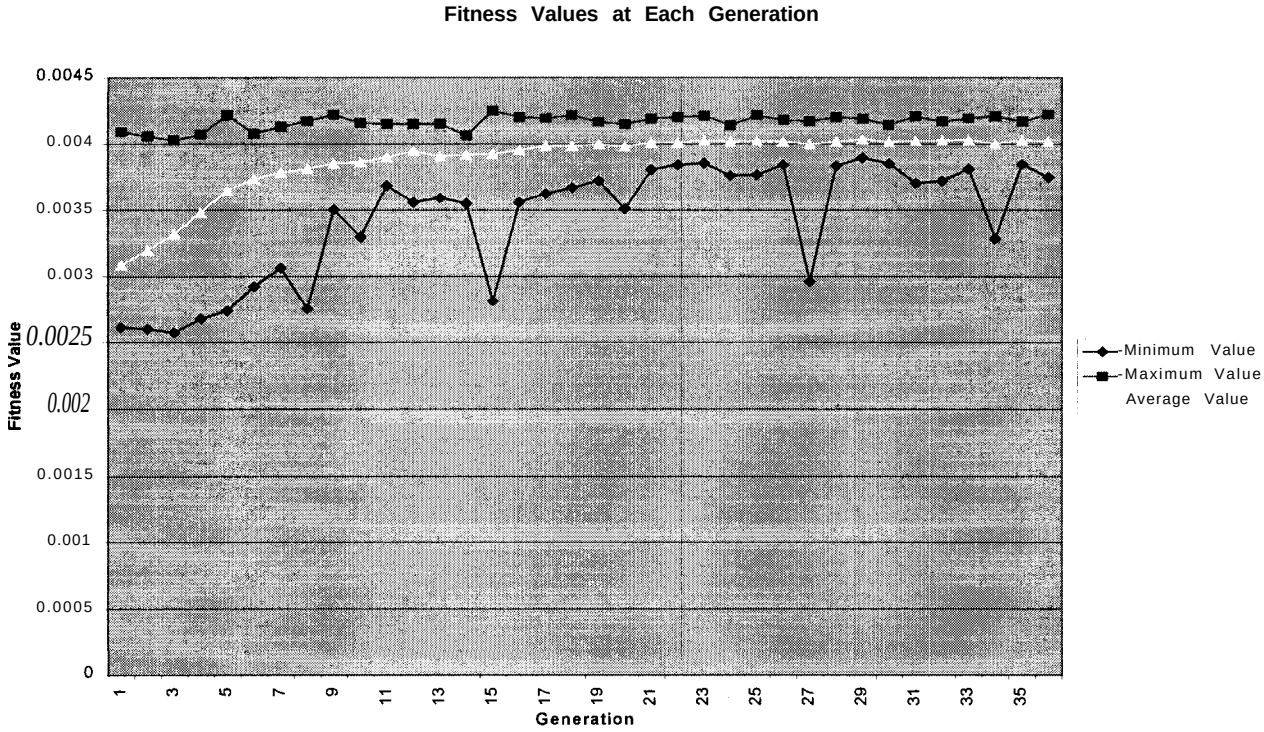


*Figure 1: Fitness values for a typical run on **eil51** (as seen in Table 1). There were no **significant** changes after the 38"' generation.*

5. Conclusions

We can conclude that this ant system, when optimized by the genetic algorithm can find the optimal solution for small problems and also, close to optimal solutions for medium problems.

What we can observe from the data is that some of the parameters seem to be more important than others. For example, in every case, the num_of_ants parameter ends up being very large. However, parameters such as pher_factor are very variable. This leads to the conclusion that num-of-ants is a more important parameter. Statistical analysis could be used to validate this and discover which parameters have the most effect.

Additionally, it can be noted that there is not a significant decrease in the best path size from generation to generation. However, this may be because there are 50 possibilities in each generation. We can observe that the average becomes better, as well as the worst case (see Figure 1). Because some of the parameters seem much more important than others, this gives each generation a good probability of getting one of those parameters right in at least one individual. It is likely that better results for the best path could be found with more limitations imposed on certain parameters (i.e. the num-of-ants parameter).

6. Future Work

Although the results of this experiment are promising, they are surely **just** the beginning. Undoubtedly, even better results would be found with an improved decision-making scheme for the "ants" and a better local optimization system for the best path. Also, it appears that some of the parameters have more effect than others. It would be interesting to try and simply optimize the "good" parameters and leave the "bad" ones out of the GA optimization. Additionally, this system seems to lend itself easily to integrating some elements of a Simulated Annealing system into it. Finally, it would be interesting to add in some elements of other AC variations [8].

Bibliography

[1] Lin, S. (1965) "Computer Solutions of the Traveling Salesman Problem." *Bell Syst. Journal* 44, 2245-2269.

[2] Lin, S., and B. W. Kernighan. (197 1) "An Effective Heuristic Algorithm for the Traveling Salesman Problem." *Oper. Res. 2* 1, 498-5 16.

[3] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M. N., Teller, A.H. and Teller, E., (1958) "Equations of State Calculations by Fast Computing Machines", *J. Chem. Phys. 21,* 1087- 1092.

[4] Matsuda, S., (1996) "Set Theoretic Comparison of Mappings of Combinatorial Optimization Problems to Hopfield Neural Networks.", *Systems and Computers in Japan* 27, 45-59

[5] Goldberg, D., *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison Wesley, 1989

[6] Mascato, P., (1989) "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts : Towards Memetic Algorithms", *Caltech Concurrent Computation program* 158-79.

[7] Dorigo, M., V. Maniezzo & A. Colorni (1991). "The Ant System: An Autocatalytic Optimizing Process". *Technical Report No. 91-016 Revised, Politecnico di Milano, Italy*

[8] Bonabeau, E., Dorigo, M. & Theraulaz, *G., Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press. 1999, 39-56

[9] Lin, F., Kao, C., & Hsu, C., (1993) "Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems". *IEEE Transactions on Systems, Man, and Cyberspace 23,* 1752-1767.

[10] Bonomi, E., & Lutton, J., (1984) "The N-city Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm". *SIAM Review, 26* (4), *55* 1 – 569.

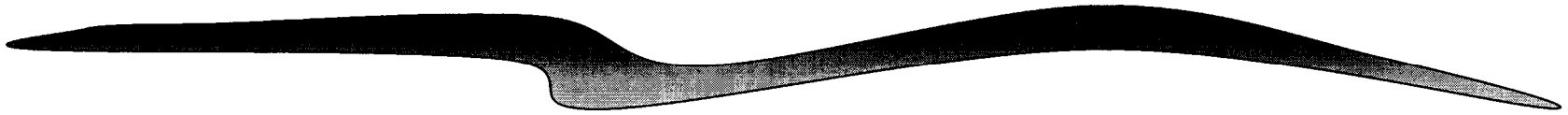
[1 1] Dorigo, M. & Gambardella, L., (1997) "Ant Colonies for the Traveling Salesman Problem". *BioSystems, 43, 73-8* 1.

[12] Mühlenbein, H., (1992) "Parallel Genetic Algorithms in Combinatorial Optimization". *Computer Science and Operations Research. 44* 1-456.

[13] http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/

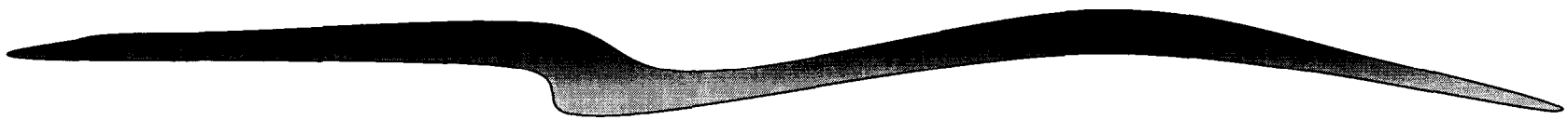
[14] http://www.keck.caam.rice.edu/tsp/usa13509/

Evolved and Timed Ants



Optimizing the Parameters of a Time-Based Ant System Approach to the Traveling Salesman Problem Using a Genetic Algorithm.

Who Am I?

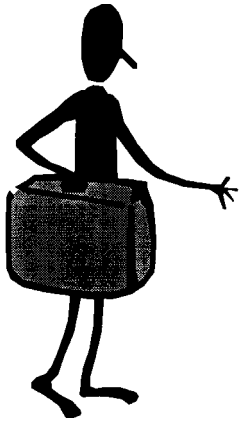


- Senior: New Mexico State University
- Computer Science Department
- Sandia Summer Intern For Three Years
- Background in Java Programming
- Background in Emergent Behavior Programming

The Project

This project looked for a solution to the Traveling Salesman Problem using an Ant System approach.

Further, it optimized the Ant System using a Genetic Algorithm.



The Traveling Salesman Problem

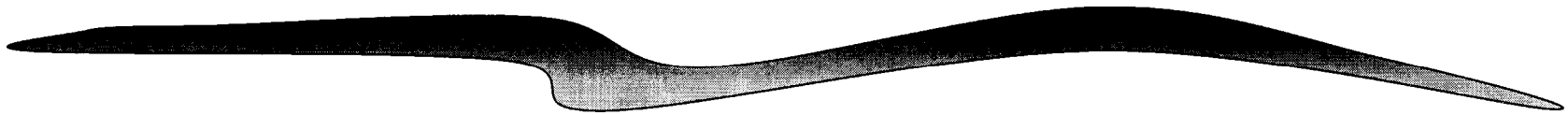
A traveling salesman must:

- Travel to many cities
- Take the quickest (shortest) route
- Only visit each city once
- End up back where he started

Number of Possible Paths

Number of Cities	Number of Paths
5	120
10	3628800
15	130767436800
20	2432902008176640000
25	15511210043330985984000000

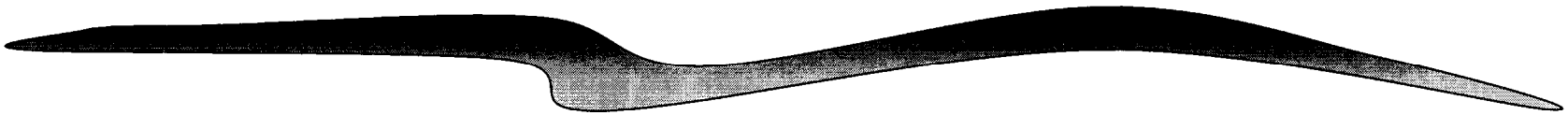
The Ant System



Based on Foraging Patterns of Real-Life Ants:

- Ants move randomly through the map
- Ants drop pheromone
- Ants choose where to go next based on amount of pheromone
- Pheromone evaporates
- Optimal solution is found by ants
- Optimal solution is improved by local search function

The Timed Ant System



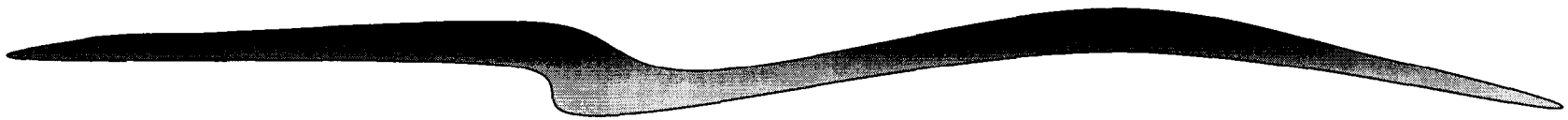
This system is based on the time taken between cities.

```
for time = 0 to max time STEP increment_value
  for i = 1 to numb& of ants
    if (ants[i] has time-left to get to the next node)
      decrement ants[i]'s time left by increment_value
    else
      pick which node to move to next, set ants[i]'s time
      left and update the pheromone on the edge just passed
```

Genetic Algorithms

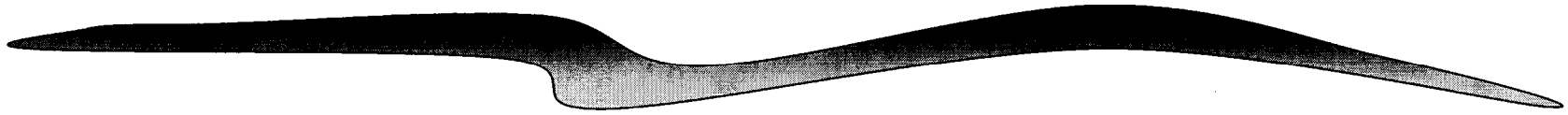
- Evolution Function
  - Creates strings of bits that encode the parameters being evolved
  - Gets fitness for strings and evolves based on fitness values
- Fitness Function
  - Receives string of bits
  - Returns fitness after running the decoded parameters through the program

The Genetic Algorithm



- Used up to 100 generations of 50 individual bit strings
- Each generation kept the best bit strings from the previous generation
- Other bit strings made by combining the good ones
- Some random changes allowed in bit strings
- Fitness found based on best path found by Ant System

Parameters Evolved



increment – 5 bits : range: [1, 32]

num_of_ants – 8 bits : range: [1, 256]

evaporation increment – 5 bits : range: [1, 32]

evaporation rate – 8 bits : range: (0, 1]

add pheromone1 – 8 bits : range: (0, 1]

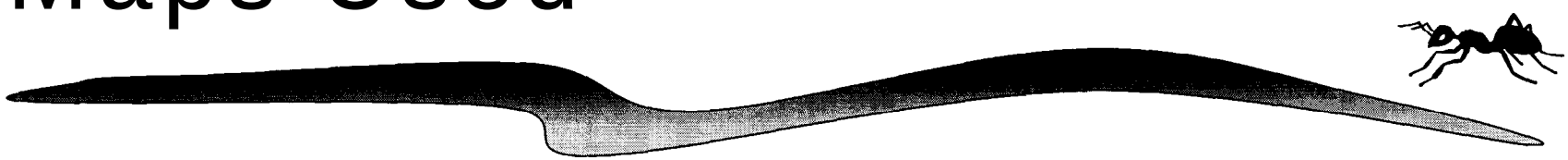
add pheromone2 – 8 bits : range: (0, 1]

dist factor – 10 bits : range: (0, 10]

pher factor – 10 bits : range: (0, 10]

rand thresh – 10 bits : range: (0, 1]

Maps Used

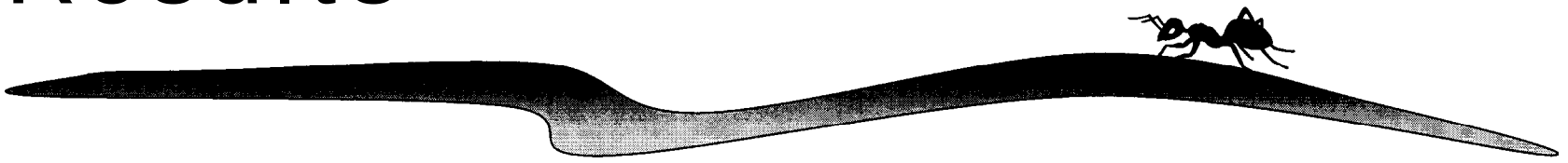


These maps were found at the TSPLIB - an online resource for Traveling Salesman Problems and solutions

Ulysses 16 - A 16 city map (small)

Ei15 1 - A 5 1 city map (medium)

Results



- Optimal solutions were found for the ulysses16 map and near optimal solutions were found for the ei15 1 map.

- Semi-optimized parameters found for Ant System for each map

Conclusions Made

- Optimizing an Ant System with a Genetic Algorithm can improve the answers found
- Some parameters seem more important than others
- It is easy to get a good answer with 50 input strings

Future Work

- Just the beginning
- Use larger maps
- Put stronger constraints on input parameters
- Emphasize difference between major and minor parameters
- Improve Ant System algorithm
- Optimize Ant System in favor of time taken as well as best path found